Workgroup C1.067

LINT report

25/05/2025

Javier Arellano López – [javarelop@alum.us.es](mailto:javarelop@alum.us.es)

https://github.com/javiarellanoo/Acme-ANS-D04

Table of Contents

[EXECUTIVE SUMMARY: 2](#_Toc199092838)

[REVISION TABLE 3](#_Toc199092839)

[INTRODUCTION 4](#_Toc199092840)

[IDENTIFIED BAD SMELLS 4](#_Toc199092841)

[Commonly repeated bad smells 4](#_Toc199092842)

[Bad smells in entities and validators 5](#_Toc199092843)

[Bad smells on services 6](#_Toc199092844)

[CONCLUSIONS 7](#_Toc199092845)

[BIBLIOGRAPHY 7](#_Toc199092846)

# EXECUTIVE SUMMARY:

This document serves as a comprehensive overview of the potential bad smells reported by Sonar’s Lint over the requirements of Student #1. The bad smells reported will be analyzed and justified in the case that they are considered innocuous. The code smells not included in this report have already been corrected.

# REVISION TABLE

|  |  |  |
| --- | --- | --- |
| Revision Number | Date | Description |
| 1.0 | 25/05/2025 | Initial version of the document |
|  |  |  |
|  |  |  |

# INTRODUCTION

The goal of this document is to provide an analysis of the bad smells identified in the code of Student #1, grouped by the different sections of the project in where they appear. A clear justification of why the code smell has been reported as innocuous will be provided for each of them.

In case that a certain code smell is repeated across different features (i.e. recommendations of defining constants), they will be assessed as one, indicating clearly where they were detected.

Finally, a conclusion on the utility of this tool will be presented.

# IDENTIFIED BAD SMELLS

## Commonly repeated bad smells

Definition of Constants:

Scattered through the code, Sonar’s Lint has recommended that constants are defined to avoid repetition of the same Strings. More specifically, it has been detected in the following places:

* Flight: “Define a constant instead of duplicating the literal “Not defined yet” 3 times.”
  + “Not defined yet” is a string employed as a place holder for the origin and destination cities in the listing of Flights without Legs. Since they do not have any impact over any feature, it has been considered that their presence is innocuous.
* ManagerFlightCreateService, ManagerFlightUpdateService, ManagerFlightPublishService: “Define a constant instead of duplicating the literal “airline” 3 times”
  + The duplicated literal is inside requests for the data of the request to avoid hacking attempts as well as to the unbinding of the Flight, even if it could be supplanted by a constant, due to the few times that it appears, it does not seem appropriate to use constants in the process of adding entries to the dataset of the response.
* ManagerLegCreateService, ManagerLegUpdateService, ManagerLegPublishService: “Define a constant instead of duplicating the literal “destinationAirport” 3 times”, “Define a constant instead of duplicating the literal “departureAirport” 3 times”, “Define a constant instead of duplicating the literal “aircraft” 3 times”
  + The same explanation as above applies to this case, but referring to the Legs of flights.
* ManagerLegCreateService: “Define a constant instead of duplicating the literal “masterId” 6 times”
  + The same explanation as above applies in this case.

## Bad smells in entities and validators

Override the “equals” method in this class

This possible bad smell is appreciated in all entities and realms of the project, since no explicit equals methods are defined for them. Since they all extend AbstractEntity or AbstractRole, this is already handled by the framework.

Remove field injection and use constructor

This possible bad smell is appreciated in all custom validators that involve the usage of a Repository. Sonar’s Lint complains about the use of the @Autowired annotations instead of creating proper constructors. This is not deemed necessary, since it would defeat the whole purpose of using annotations to avoid the creation of trivial code.

Replace assert by proper check

Sonar’s Lint complains about the usage of “assert context != null” type of statement in all validators. Since the context is provided by the framework, this does not fall under our scope of operability.

Remove this use of “getYear”; it is deprecated.

Sonar’s Lint advises to use a different approach as to obtain the year of a Date object. However, this method is still available so that the code smell does not hide any bugs but it is rather a warning that in possible future versions, this method will not be supported.

## Bad smells on services

Remove field injection and use constructor

This possible bad smell is appreciated in all services. Sonar’s Lint complains about the use of the @Autowired annotations instead of creating proper constructors. This is not deemed necessary, since it would defeat the whole purpose of using annotations to avoid the creation of trivial code.

Remove empty statement

Sonar’s Lint complains about the presence of empty validate methods in some services. Since no custom validations need to be performed at feature level, but rather at entity level in said cases, duplicating the logic defined on the corresponding entity’s validator would be counterproductive.

Add a nested comment explaining why this method is empty, throw an UnsupportedOperationException or complete the implementation.

Sonar’s Lint complains about the same empty methods defined above, in this case, as explained before, the logic inside would be duplicated, hence, it does not make sense to complete the implementation or to throw an Exception. As studied in other subjects, clean code does not require comments to be understood, hence said comment has not been added.

Use a primitive boolean expression

Sonar’s Lint suggests to change the value of the expression “!leg.getDraftMode()” from a Boolean type to boolean, in order to avoid possible null values. Since this values have no way to be illegally changed, they will never be null so that it is not worth the change.

# CONCLUSIONS

Sonar’s Lint has been a useful and powerful tool to identify potential code smells in our code without the need of running Eclipse’s coverage, identifying risks without compiling the project and reproducing test cases. However, it is not perfect since it will analyze the code based on common patterns and errors. It also provides the option to detect the aforementioned potential smells “on the fly” so that it is definitely a useful tool to keep in our future journey as developers.

# BIBLIOGRAPHY

Intentionally blank.